

Refining the Analysis of Divide and Conquer: How and When

Jérémy Barbay¹, Carlos Ochoa^{1*}, and Pablo Pérez-Lantero²

¹ Departamento de Ciencias de la Computación, Universidad de Chile, Chile
jeremy@barbay.cl, cochoa@dcc.uchile.cl.

² Escuela de Ingeniería Civil Informática, Universidad de Valparaíso, Chile.
pablo.perez@uv.cl.

Abstract. Divide-and-conquer is a central paradigm for the design of algorithms, through which some fundamental computational problems, such as sorting arrays and computing convex hulls, are solved in optimal time within $\Theta(n \log n)$ in the worst case over instances of size n . A finer analysis of those problems yields complexities within $O(n(1 + \mathcal{H}(n_1, \dots, n_k))) \subseteq O(n(1 + \log k)) \subseteq O(n \log n)$ in the worst case over all instances of size n composed of k “easy” fragments of respective sizes n_1, \dots, n_k summing to n , where the entropy function $\mathcal{H}(n_1, \dots, n_k) = \sum_{i=1}^k \frac{n_i}{n} \log \frac{n}{n_i}$ measures the “difficulty” of the instance. We consider whether such refined analysis can be applied to other algorithms based on divide-and-conquer, such as polynomial multiplication, input-order adaptive computation of convex hulls in 2D and 3D, and computation of Delaunay triangulations.

Keywords: Adaptive Analysis, Convex Hull, Delaunay Triangulation, Divide and Conquer, Voronoi Diagrams.

1 Introduction

The divide-and-conquer paradigm is used to solve central computational problems such as SORTING arrays [10], computing CONVEX HULLS [9], DELAUNAY TRIANGULATIONS and VORONOI DIAGRAMS [6] of points in the plane and in higher dimensions, MATRIX MULTIPLICATION [19], POLYNOMIAL MULTIPLICATION [16], HALF-PLANE INTERSECTION [15], among others. For some of these problems, this paradigm yields an optimal running time within $\Theta(n \log n)$ in the worst case over instances of n elements.

An adaptive analysis of slight variants of some of these algorithms yields improved running times on large classes of instances. Those results can be refined [1,2,14] up to complexities within $O(n(1 + \mathcal{H}(n_1, \dots, n_k))) \subseteq O(n(1 + \log k)) \subseteq O(n \log n)$ in the worst case over all instances of size n composed of k “easy” fragments of respective sizes n_1, \dots, n_k summing to n , where the entropy function $\mathcal{H}(n_1, \dots, n_k) = \sum_{i=1}^k \frac{n_i}{n} \log \frac{n}{n_i}$ measures the “difficulty” of the instance. We describe here two examples for the problem of SORTING arrays (see Section 2 for more).

- Munro and Spira [14] showed that the algorithm **MergeSort** can be adapted to sort a multiset S of n elements in time within $O(n(1 + \mathcal{H}(m_1, \dots, m_\sigma))) \subseteq O(n(1 + \log \sigma)) \subseteq O(n \log n)$, where σ is the number of distinct elements in S and m_1, \dots, m_σ (such that $\sum_{i=1}^\sigma m_i = n$) are the ρ multiplicities of the distinct elements in S , respectively.
- Taking advantage of the input order, Knuth [10] considered sequences formed by *runs* i.e., contiguous increasing subsequences, and described an algorithm sorting such sequences in time within $O(n(1 + \log \rho)) \subseteq O(n \lg n)$, where ρ is the number of *runs* in the sequence. Barbay and Navarro [2] improved the analysis of this algorithm in time within $O(n(1 + \mathcal{H}(r_1, \dots, r_\rho))) \subseteq O(n(1 + \log \rho)) \subseteq O(n \log n)$, where r_1, \dots, r_ρ (such that $\sum_{i=1}^\rho r_i = n$) are the sizes of these *runs*.

* Corresponding author.

² Similar analysis techniques have been applied to some other problems, though only partially. Following an example for the problem of computing the CONVEX HULL in the plane (see Section 3.2 for more): Levkopoulos et al. [11] described an adaptive algorithm for computing the CONVEX HULL of a polygonal chain. The algorithm takes advantage of the minimum number κ of simple subchains into which the polygonal chain can be partitioned. They showed that the time complexity of the algorithm is within $O(n(1+\log \kappa)) \subseteq O(n \log n)$.

Hypothesis. Which similar refinements can be applied to which divide-and-conquer algorithms, if any, and to what depth?

Our Results. In Section 2, we list and classify previous refined analyses between those that are *Structure Based* and those *Input-Order Based*. In Section 3.1, we describe a refined analysis of the principal step in the algorithm for POLYNOMIAL MULTIPLICATION using the *Fast Fourier Transformation*. In Sections 3.2 and 3.3, we describe two distinct refined analyses for problems in computational geometry, which yield various optimal input-order adaptive results on the computation of CONVEX HULLS, DELAUNAY TRIANGULATIONS, and VORONOI DIAGRAMS in the plane. In Section 3.2, we refine the analysis of Levkopoulos et al. [11]’s algorithm for computing the CONVEX HULL of polygonal chains in time within $O(n(1 + \mathcal{H}(n_1, \dots, n_\kappa))) \subseteq O(n(1 + \log \kappa)) \subseteq O(n \log n)$, where n_1, \dots, n_κ are the lengths of the subchains of a partition of a polygonal chain of n points into the minimum number κ of simple subchains. In Section 3.3, we describe a refined analysis of the computation of VORONOI DIAGRAMS and DELAUNAY TRIANGULATIONS for sequences S formed by n points, which yields a time complexity within $O(n(1 + \mathcal{H}(v_1, \dots, v_\mu))) \subseteq O(n(1 + \log \mu)) \subseteq O(n \log n)$, where v_1, \dots, v_μ are the sizes of the minimum number μ of monotone histograms in which S can be cut, with respect to two fixed orthogonal directions and show, as a corollary, an upper bound for computing the CONVEX HULL of a sequence S formed by n points in time within $O(n(1 + \mathcal{H}(v_1, \dots, v_\mu))) \subseteq O(n(1 + \log \mu)) \subseteq O(n \log n)$, where v_1, \dots, v_μ are the sizes of the minimum number μ of monotone histograms in which S can be cut, with respect to two fixed orthogonal directions. In Section 4, we describe some more difficult applications of such refined analyses. In Section 5, we discuss the possibility of designing algorithms which analyses combine *Structure Based* and *Input-Order Based* results synergically as opposed to running them in parallel.

2 Classification of Results

We review here some results on the refined analysis of algorithms for SORTING arrays and for computing planar CONVEX HULLS, DELAUNAY TRIANGULATIONS and VORONOI DIAGRAMS. We classify the various refined analysis between those focusing on the structure of the instance (Section 2.1) versus those focusing on the order in which the input is given (Section 2.2). There does not seem to be any example where both strategies are mixed: we discuss the potential for those in Section 5.

2.1 Structure Based Results

By “Structure Based Results” we mean algorithms taking advantage of the structure of the instance, for example, taking advantage of the frequencies of the values in a multiset or of the relative positions of the points in a set. Such results are known for SORTING multisets and for computing CONVEX HULLS.

MergeSort is a divide-and-conquer SORTING algorithm in the comparison model. This algorithm relies on a linear time merge process, that combines two ordered sequences into a single ordered sequence.

Concerning the problem of SORTING, Munro and Spira [14] considered the task of SORTING a multiset $S = \{x_1, \dots, x_n\}$ of n real numbers with σ distinct values, of multiplicities m_1, \dots, m_σ , respectively, so that $\sum_{i=1}^{\sigma} m_i = n$. They showed that adding counters to various classical algorithms (among which the divide-and-conquer based algorithm **MergeSort**) yields a time complexity within $O(n(1 + \mathcal{H}(m_1, \dots, m_\sigma))) \subseteq O(n(1 + \log \sigma)) \subseteq O(n \log n)$ for SORTING a multiset, where $\mathcal{H}(m_1, \dots, m_\sigma) = \sum_{i=1}^{\sigma} \frac{m_i}{n} \log \frac{n}{m_i}$ measures the entropy of the distribution of the multiplicities $\langle m_1, \dots, m_\sigma \rangle$. This result takes advantage of the frequencies of the values i.e., the structure of the instance.

Given a set P of n points, the *Convex Hull* of P is the smallest convex set containing P (see Figure 1)³. Considering the problem of computing the CONVEX HULL, Kirkpatrick and Seidel [9] described an algorithm to compute the CONVEX HULL of a set of n planar points in time within $O(n(1 + \log h)) \subseteq O(n \log n)$, where h is the number of vertices in the CONVEX HULL. The algorithm relies on a variation of the divide-and-conquer paradigm, which they call the “Marriage-Before-Conquest” principle. For computing the upper hull, the algorithm finds a vertical line that divides the input point set into two approximately equal-size parts in linear time. Next, it determines the edge of the upper hull that intersects this line in linear time. It then eliminates the points that lie underneath this edge and finally applies the same procedure to the two sets of the remaining points on the left and right side of the vertical line. A similar algorithm computes the lower hull. Afshani et al. [1] refined the complexity analysis of this algorithm to within $O(n(1 + \mathcal{H}(n_1, \dots, n_h))) \subseteq O(n(1 + \log h)) \subseteq O(n \log n)$, where n_1, \dots, n_h are the sizes of a partition of the input, such that every element of the partition is a singleton or can be enclosed by a triangle whose interior is completely below the upper hull of the set, and $\mathcal{H}(n_1, \dots, n_h)$ has the minimum possible value (minimum entropy of the distribution of the points into a certificate of the instance). This result takes advantage of the positions of the points i.e., the structure of the instance.

2.2 Input-Order Based Results

By “Input-Order Based Results” we mean algorithms taking advantage of the order of the input, for example, taking advantage of the order of the values in a sequence of numbers or of the order in which the points are given in a polygonal chain. We describe only a sampling of such results on the problem of SORTING permutations (see the survey from Moffat and Petersson [13] for more), revisit some results on the computation of CONVEX HULLS in the plane and 3D space, and show that those results in the plane are actually only “Input-Order Based”. Those results for computing CONVEX HULLS in 3D space show that no algorithm can take advantage of the position of the points i.e., structure based, in order to compute DELAUNAY TRIANGULATIONS in the plane.

Concerning the problem of SORTING, Knuth [10] described an adaptive sorting algorithm that takes advantage of permutations formed by sorted blocks called *runs*, that is, subsequences of consecutive positions in the input with a positive gap between successive values, from beginning to end. He showed that the time complexity of this algorithm is within $O(n(1 + \log \rho)) \subseteq O(n \lg n)$, where ρ is the number of *runs* in the permutation (e.g. $(1, 2, 6, 7, 8, 9, 3, 4, 5)$ is composed of 2 such sorted blocks $(1, 2, 6, 7, 8, 9)$ and $(3, 4, 5)$). Barbay and Navarro [2] refined the algorithm described by Knuth [10]. They included in the analysis not only the number of *runs* but also their sizes. The main idea is to detect the runs first and then merge them pairwise, using a mergesort-like step. The detection of ascending runs can be done in linear time by a scanning process identifying the positions i in π such that $\pi(i) > \pi(i + 1)$. Merging the two shortest runs at each step further reduces the number of comparisons, making the running time of the merging process adaptive to the entropy of the sequence of the lengths of the runs. The merging process is then represented by a tree with the shape of a Huffman [7] tree, built from the distribution of the *runs* sizes. They extend this result to mix ascending and descending runs, showing that, if the permutation π is formed by ρ runs of sizes given by the vector $\langle r_1, \dots, r_\rho \rangle$, then π can be sorted in time within $O(n(1 + \mathcal{H}(r_1, \dots, r_\rho))) \subseteq O(n(1 + \log \rho)) \subseteq O(n \log n)$. This result takes advantage of the order of the values in the input i.e., the input order.

Considering the computation of the CONVEX HULL in the plane, Levkopoulos et al. [11] described a divide-and-conquer algorithm for computing the CONVEX HULL of a polygonal chain. The algorithm is based in the fact that the CONVEX HULL of a simple chain can be computed in linear time, and that deciding whether a given chain is simple can be done in linear time. They measured the complexity of this algorithm in terms of the minimum number of simple subchains κ into which the chain can be cut. They showed that the time complexity of this algorithm is within $O(n(1 + \log \kappa)) \subseteq O(n \log n)$. We improve the analysis of this algorithm including not only the minimum number of simple subchain into which the polygonal chain can be partitioned but also their sizes (see Section 3.2). This result takes advantage of the order in which the points are given i.e., the input order.

Given a set P of n planar points, a *triangulation* of P is a subdivision of the convex hull of P into triangles with vertex set the set P . Concerning the computation of the CONVEX HULL in 3D, a related

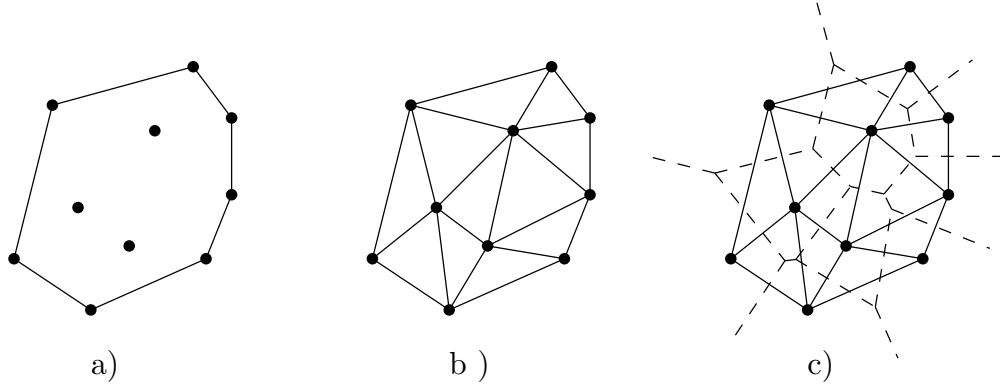


Fig. 1. A point set P . a) The convex hull of P , b) the Delaunay triangulation of P , and c) the Delaunay triangulation and the Voronoi diagram of P .

concept is that of the DELAUNAY TRIANGULATION $DT(P)$ of a point set P in the plane, a triangulation where for every edge e there exists a disk C with the following properties: (i) the endpoints of edge e are on the boundary of C , and (ii) no other point of P is in the interior of C : it is named after Boris Delaunay for his work on this topic in 1934. An equivalent definition is such that no point in P is inside the circumcircle of any triangle of $DT(P)$. Computing the DELAUNAY TRIANGULATION is equivalent to computing its dual, called the VORONOI DIAGRAM: each one can be constructed from the other in linear time [15] (see Figure 1). Computing the DELAUNAY TRIANGULATION of a set of points in two dimensions reduces to computing the CONVEX HULL in three dimensions of the projections of those points on an hyperbolic plane. The projection of P onto the unit elliptic paraboloid $z = x^2 + y^2$ yields a point set P' . The CONVEX HULL $CH(P')$ of P' contains every point of the set. The downward-facing facet of $CH(P')$ are those whose normal vectors have a negative z -value. Projecting the edges of downward-facing facet in $CH(P')$ onto the plane yields the DELAUNAY TRIANGULATION of P . By showing tight bounds for input-order oblivious i.e., structure based, algorithms for computing CONVEX HULLS in three dimensions, Afshani et al. [1] indirectly proved that no planar DELAUNAY TRIANGULATION algorithm can take advantage of the position of the points.

Theorem 1 (Afshani et al. [1]). *Consider a set of n points in the plane. For any algorithm A computing the Delaunay triangulation in the algebraic decision tree model, A performs in time within $\Omega(n \log n)$ on average on a random order of the points. This implies that there is an order of those points for which A performs in time within $\Omega(n \log n)$.*

We describe in Sections 3 and 4 some algorithms taking advantage of the order of the input to compute DELAUNAY TRIANGULATIONS and VORONOI DIAGRAMS, among other desirable objects in computational geometry.

3 Refined Analysis: Three Examples

We show in this section how to refine the analysis of the principal step in the algorithm for multiplying polynomials using the *Fast Fourier Transformation* [16], how to refine the analysis of the algorithm from Levkopoulos et al. [11] for the decomposition of a polygonal chain into simple sequences, and how to extend this analysis to the computation of VORONOI DIAGRAMS and DELAUNAY TRIANGULATIONS for another measure of difficulty based on monotone histograms.

3.1 Polynomial Multiplication: Adaptivity to Zero-coefficients

Given two polynomials $A = (a_0, \dots, a_{n-1})$ and $B = (b_0, \dots, b_{n-1})$ described by their coefficients, the polynomial multiplication problem is to compute the coefficients of the polynomial $C = A \cdot B$. The approach

to multiplying polynomials using the *Fast Fourier Transformation* [16] can be divided into three steps: (i) evaluate the polynomials A and B in $2n$ values (the $(2n)$ th roots of the unity); (ii) evaluate C in these $2n$ values by multiplying the evaluations of A and B ; and (iii) obtain the coefficients of C by interpolation using the values computed in the step (ii). The steps (i) and (iii) are accomplished by a divide-and-conquer algorithm for polynomial evaluation. We refine the analysis of the divide-and-conquer polynomial evaluation algorithm to take advantage of the number of zero-coefficients and of their relative positions in the vector of coefficients that describes the polynomial.

Given a polynomial A , the polynomial evaluation algorithm defines two polynomials, A_{even} and A_{odd} , that consist of the even-indexed and odd-indexed coefficients of A , respectively. Hence, $A(x) = A_{\text{even}}(x^2) + x \times A_{\text{odd}}(x^2)$. If x is one of the $(2n)$ th roots of the unity, then x^2 is one of the n th roots of the unity. In order to evaluate the polynomial A on each of the $(2n)$ th roots of the unity, the recursive procedure divides A into A_{even} and A_{odd} , evaluates A_{even} and A_{odd} in the n th roots of the unity, and once these values are computed, evaluate A on each of the $(2n)$ th roots of the unity using the formula $A(x) = A_{\text{even}}(x^2) + x \times A_{\text{odd}}(x^2)$. The time complexity $T(n)$ of this algorithm follows the recurrence $T(n) \leq T(\frac{n}{2}) + O(n)$, which yields a time complexity within $O(n \log n)$. If at one step of the recursion call all the coefficients are zero, the algorithm finishes the computation at this branch (see Figure 2).

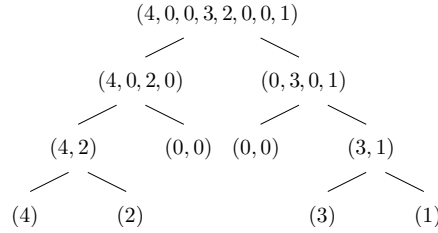


Fig. 2. Vector formed by the coefficients of the polynomial $A(x) = 4 + 3x^3 + 2x^4 + x^7$ and the recursion tree of the divide-and-conquer evaluation algorithm. At each step, the algorithm divides the coefficients of A into A_{even} and A_{odd} .

Given a polynomial $A = (a_0, \dots, a_{n-1})$, we define the equivalence relation E between the positions of the zero-coefficients in A : two positions of zeros p and q are equivalent if and only if (i) there exists $k \in \mathbb{Z}^+$ such that $p \equiv q \pmod{2^k}$; (ii) all the positions r such that $r \equiv p \pmod{2^k}$ are zeros; and (iii) there exist a position t such that $t \equiv p \pmod{2^{k-1}}$ and the value in t is different from zero. The idea of this equivalence relation is to group the positions in classes such that the positions in the same class form a vector where all the coefficients are zero in a node of the recursion tree. Let ζ and η be the number of zeros in the vector formed by the coefficients of A and the number of equivalence classes defined by E , respectively. Let $\langle n_1, \dots, n_\eta \rangle$ be the vector formed by the sizes of the equivalence classes defined by E . Then, $\sum_{i=1}^{\eta} n_i = \zeta$. The following theorem sets the refined analysis in function of ζ and the vector $\langle n_1, \dots, n_\eta \rangle$.

Theorem 2. *The complexity of the evaluation algorithm for polynomials of n coefficients, number of zero-coefficients ζ and vector $\langle n_1, \dots, n_\eta \rangle$ formed by the sizes of the equivalence classes defined by E is within $O((n - \zeta) \log n + \sum_{i=1}^{\eta} n_i \log \frac{n}{n_i}) \subseteq O(n \log n)$.*

In the following sections, we apply similar techniques to obtain optimal refined analysis for input-order adaptive algorithms computing CONVEX HULLS and DELAUNAY TRIANGULATIONS in the plane.

3.2 Computing Convex Hulls: Adaptivity to Simple Subchains

A *polygonal chain* is a curve specified by a sequence of points p_1, p_2, \dots, p_n . The curve itself consists of the line segments connecting the pairs of consecutive points. A polygonal chain C is *simple* if any two edges of C that are not adjacent are disjoint, or if the intersection point is a vertex of C ; and any two adjacent edges

share only their common vertex. Melkman [12] described an algorithm that computes the CONVEX HULL of a simple polygonal chain in linear time, and Chazelle [3] described an algorithm for testing whether a polygonal chain is simple in linear time.

Levcopoulos et al. [11] combined these results to yield an adaptive divide-and-conquer algorithm for computing the CONVEX HULL of polygonal chains. The algorithm tests if the chain C is simple, using Chazelle [3]’s algorithm: if the chain C is simple, the algorithm computes the CONVEX HULL of C in linear time, using Melkman [12]’s algorithm. Otherwise, if C is not simple, the algorithm cuts C into the subsequences C' and C'' , whose sizes differ at most in one; recurses on each of them; and merges the resulting CONVEX HULLS using Preparata and Shamos’s algorithm [15]. They measured the complexity of this algorithm in terms of the minimum number of simple subchains κ into which the chain C can be cut. Let $t(n, \kappa)$ be the worst-case time complexity taken by this algorithm for an input chain of n vertices that can be cut into κ simple subchains. They showed that $t(n, \kappa)$ satisfies the following recursion relation: $t(n, \kappa) \leq t(\lceil \frac{n}{2} \rceil, \kappa_1) + t(\lfloor \frac{n}{2} \rfloor, \kappa_2)$, $\kappa_1 + \kappa_2 \leq \kappa + 1$. The solution to this recursion gives $t(n, \kappa) \in O(n(1 + \log \kappa)) \subseteq O(n \log n)$. In the sequel, this algorithm will be named as **Test-And-Divide**.

The **Test-And-Divide** algorithm partitions the input chain into simple subchains. If it was possible to partition the input chain into the minimum number κ of simple subchains in linear time, then the same approach described by Barbay and Navarro [2] could be applied to obtain a refined analysis in function of $O(n(1 + \mathcal{H}(n_1, \dots, n_\kappa))) \subseteq O(n(1 + \log \kappa)) \subseteq O(n \log n)$. But, as far as we know, there does not exist any linear time algorithm to accomplish this task.

In the recursion tree of the execution of the **Test-And-Divide** algorithm on input C of n points, every node represents a subchain of C . The cost of every node is linear in the size of the subchain that it represents. The simplicity test and the merge process are both linear in the number of points in the subchain. When this subchain is simple the node that represents this subchain is a leaf. Every time this algorithm discovers that the polygonal chain is simple, it executes a number of operations linear in the size of the chain and the corresponding node in the recursion tree becomes a leaf.

Width Analysis: A Warm-up. Consider for illustration the particular case of a polygonal chain C of $n = 2^m$ planar points such that C can be partitioned into the minimum number of simple subchains $\kappa = m$ of lengths $2^1, 2^1, 2^2, 2^3, \dots, 2^{m-1}$, respectively. Every time the algorithm cuts the current chain in half, the right subchain is simple and the recursive call is made only in the left subchain. Hence, the recursion tree of the algorithm on input C has only two nodes per level, one of which is a leaf (see Figure 3). The overall running time of the algorithm on C is then within $O(n)$.

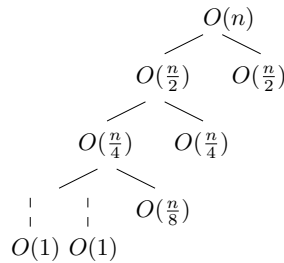


Fig. 3. The recursion tree of A on C . Each node represents a recursive call. Noted in each node is the asymptotic complexities of the simplicity test and the merging process on the subchain that it represents.

Definition 1 (Width). The width ω of the recursion tree in the execution of the **Test-And-Divide** algorithm on input C is the maximum number of nodes at any level.

Levcopoulos et al. [11] analyzed the complexity of this algorithm in the worst case over instances of fixed size n and κ . The following lemma gives an alternate analysis in the worst case over instances of fixed size n and width ω .

Lemma 1. *Let ω be the width of the recursion tree in the execution of the **Test-And-Divide** algorithm on input C of n planar points. The complexity of this algorithm on input C is within $O(\omega n)$.*

Refined Analysis. Let $\langle \ell_1, \dots, \ell_m \rangle$ be the vector formed by the sizes of the subchains represented by the m leaves of the recursion tree of the **Test-And-Divide** algorithm on input C (such that $\sum_{i=1}^m \ell_i = n$). The number of operations “saved” by the algorithm every time it discovers a leaf of size ℓ_i is within $\Omega(\ell_i \log \ell_i)$ (the cost of the subtree of the perfect binary tree rooted in a node of size ℓ_i minus $O(\ell_i)$ (the operations in the leaf are not saved)). The time complexity $T(C)$ of this algorithm on input C is within $O(n \log n)$ (the cost of the perfect binary tree) minus $\Omega(\sum_{i=1}^m \ell_i \log \ell_i - \ell_i)$ (the number of operations saved by the algorithm). So, $T(C) \subseteq O(n \log n - \sum_{i=1}^m (\ell_i \log \ell_i - \ell_i)) = O(n(1 + \mathcal{H}(\ell_1, \dots, \ell_m))) \subseteq O(\omega n) \cap O(n \log m) \subseteq O(n \log n)$.

The following lemma summarizes this finer analysis of the **Test-And-Divide** algorithm in function of the width and the number of leaves in the recursion tree.

Lemma 2. *Let $\langle \ell_1, \dots, \ell_m \rangle$ be the vector formed by the sizes of the subchains represented by the m leaves of the recursion tree in the execution of the algorithm A on input C of $n = \sum_{i=1}^m \ell_i$ planar points. Let ω be the maximum width of the recursion tree. The time complexity of A on C is within $O(n(1 + \mathcal{H}(\ell_1, \dots, \ell_m))) \subseteq O(\omega n) \cap O(n \log m) \subseteq O(n \log n)$.*

Is there a relationship between the vector $\langle \ell_1, \dots, \ell_m \rangle$ formed by the sizes of the subchains represented by the leaves of the recursion tree in the execution of the **Test-And-Divide** algorithm from on input C and the vector $\langle n_1, \dots, n_\kappa \rangle$ formed by the sizes of a partition of C into κ simple subchains?

For a given polygonal chain, there can be several partitions into simple subchains of minimum size κ for it. The Levcopoulos et al.’s analysis is in the worst case over instances for n and κ fixed. We describe below a refined analysis which takes into account the relative imbalance between the sizes of the subchains. The idea behind the refinement is to bound the number of operations that the algorithm executes for every simple subchain. This analysis makes it possible to identify families of instances where the complexity of the algorithm is linear even though the number of simple subchains into which the chain is split is logarithmic.

Theorem 3. *Let $\langle n_1, \dots, n_\kappa \rangle$ be the vector formed by the sizes of the subchains of any partition Π of the chain C into the minimum number κ of simple subchains. The time complexity of the **Test-And-Divide** algorithm on input C is within $O(n(1 + \mathcal{H}(n_1, \dots, n_\kappa))) \subseteq O(n(1 + \log \kappa)) \subseteq O(n \log n)$, which is worst-case optimal in the comparison model over instances of n points that can be partitioned into κ simple subchains of sizes $\langle n_1, \dots, n_\kappa \rangle$.*

Proof. Fix the subchain c_i of size n_i in Π . In the worst case, the algorithm considers the n_i points of c_i for the simplicity test, and the merging process in all the levels of the recursion tree from the first level to the level $\lceil \log \frac{n}{n_i} \rceil + 1$, because the sizes of the subchains in these levels are greater than n_i . In the next level, one of the nodes ℓ of the recursion tree fits completely inside c_i and therefore it becomes a leaf. Hence, at least $\frac{n_i}{4}$ points from c_i are dismissed for the following iterations. The remaining points of c_i are in the left or the right ends of subchains represented by nodes in the same level of ℓ in the recursion tree. In all of the following levels, the number of operations of the algorithm involving points from c_i can be bounded by the size of the subchains in those levels. So, the sum of the number of these operations in these levels is within $O(n_i)$. As a result, the number of operations of the algorithm involving points from c_i is within $O(n_i \log \frac{n}{n_i} + n_i)$. In total, the time complexity of the algorithm is within $O(n + \sum_{i=1}^{\kappa} n_i \log \frac{n}{n_i}) = O(n(1 + \mathcal{H}(n_1, \dots, n_\kappa))) \subseteq O(n(1 + \log \kappa)) \subseteq O(n \log n)$.

We prove the optimality of this complexity by giving a tight lower bound. Barbay and Navarro [2] showed a lower bound of $\Omega(n(1 + \mathcal{H}(r_1, \dots, r_\rho)))$ in the comparison model for SORTING a sequence of n numbers, in the worst case over instances covered by ρ runs (increasing or decreasing) of lengths r_1, \dots, r_ρ , respectively, summing to n . The SORTING problem can be reduced in linear time to the problem of computing the

CONVEX HULLS of a chain of n planar points that can be cut into ρ simple subchains of lengths r_1, \dots, r_ρ , respectively. For each real number r , this is done by producing a point with (x, y) -coordinates (r, r^2) . The ρ runs (alternating increasing and decreasing) are transformed into ρ simple subchains of the same lengths. The sorted sequence of the numbers can be obtained from the CONVEX HULL of the points in linear time. \square

3.3 Computing Delaunay Triangulations: Adaptivity to Monotone Histograms

We propose a new input-order based algorithm for computing the DELAUNAY TRIANGULATION and the VORONOI DIAGRAM of a set of n planar points. A *monotone histogram* is a sequence of points sorted with respect to two orthogonal directions. The algorithm takes advantage of the minimal number μ of monotone histograms and their sizes into which a polygonal chain can be cut. A monotone histogram is also a simple polygonal chain. Therefore, an algorithm for computing CONVEX HULLS adaptive to the decomposition of the input into monotone histograms is obtained as a corollary of Theorem 3. We extend the refined analysis to the computation of DELAUNAY TRIANGULATIONS and VORONOI DIAGRAMS adaptive to the decomposition of the input into monotone histograms.

Djidjev and Lingas [4] described an algorithm which, given a monotone histogram, computes the VORONOI DIAGRAM (and hence the DELAUNAY TRIANGULATION) of the input sequence in linear time. This algorithm suggests a way to partition the input into subsequences such that the DELAUNAY TRIANGULATION of each subsequence can be computed in linear time in its length. The partitioning algorithm cuts the sequence into μ monotone histograms with respect to two orthogonal directions d_1 and d_2 . The first two points of the subsequence determine the ordering defined by the combination of ascending and descending with respect to d_1 and d_2 . Given a sequence of points and two orthogonal directions, it is possible to test whether the sequence is a monotone histograms with respect to these two directions in linear time.

Binary Merge of Voronoi Diagrams. Kirkpatrick [8] described a linear time algorithm for the merging of two arbitrary VORONOI DIAGRAMS. Given the VORONOI DIAGRAMS of two disjoint point sets P and Q , the algorithm finds the VORONOI DIAGRAM of $P \cup Q$ in time within $O(|P| + |Q|)$. The plane is partitioned into points closer to P , points closer to Q , and points equidistant from P and Q . The points equidistant from P and Q are defined as the *contour* separating P and Q . The *contour* is composed of straight line segments: it is formed from the edges of the VORONOI DIAGRAM of $P \cup Q$ that separates the points in P from the points in Q . Inside the region of points closer to P (resp. Q) the VORONOI DIAGRAM of $P \cup Q$ and the VORONOI DIAGRAM of P (resp. Q) are identical. Thus, the merging of two VORONOI DIAGRAMS can be seen as the process of cutting the VORONOI DIAGRAMS of P and Q along the contour.

This leads to a divide-and-conquer algorithm for constructing the VORONOI DIAGRAM of a set of n points and hence for computing the DELAUNAY TRIANGULATION in time within $O(n \log n)$. This time complexity of $O(n \log n)$ is asymptotically optimal in the comparison model in the worst case over instances composed of n points. Shamos and Hoey [17] showed that the construction of any triangulation over n points requires $\Omega(n \log n)$, as SORTING can be reduced to computing the triangulation of $n + 1$ points, which yields an asymptotic computational lower bound of $\Omega(n \log n)$ in the worst case over sets of n planar points, in the comparison model.

Multiary Merge. Given μ DELAUNAY TRIANGULATIONS of sizes v_1, \dots, v_μ , respectively, to be merged, we make a sequence of binary merges, reducing at each step the number of DELAUNAY TRIANGULATIONS to be merged by 1. The merging process can be represented by a binary tree where the internal nodes are the merged DELAUNAY TRIANGULATIONS, and the leaves are the original μ DELAUNAY TRIANGULATIONS. Merging the two DELAUNAY TRIANGULATIONS of minimum sizes at each step, similar as the Huffman code algorithm [7], further improves the merging process, which takes advantage of the potential disequilibrium in the distribution of the points between the μ DELAUNAY TRIANGULATIONS. We can apply the Huffman [7] algorithm to the vector $\langle v_1, \dots, v_\mu \rangle$, thus obtaining a Huffman-shaped tree representing the merging process.

Lemma 3 (Multiary Merge). *Given μ DELAUNAY TRIANGULATIONS of respective sizes $\langle v_1, \dots, v_\mu \rangle$ summing to $n = \sum_{i=1}^\mu v_i$, there is an algorithm computing the DELAUNAY TRIANGULATION of the n points in time within $O(n(1 + \mathcal{H}(v_1, \dots, v_\mu))) \subseteq O(n(1 + \log \mu)) \subseteq O(n \log n)$.*

Proof. The algorithm follows the same steps as the algorithm suggested by Huffman [7] on a set of ρ messages of probabilities $\{r_i/n\}_{i \in [1..\rho]}$:

1. Initialize a heap H with the ρ VORONOI DIAGRAMS, indexed by their size;
2. While H contains more than one VORONOI DIAGRAM
 - extract the two smallest VORONOI DIAGRAMS from H , of respective sizes n_1 and n_2 ,
 - merge them into a VORONOI DIAGRAM T of size $n_1 + n_2$, and
 - insert T in H .

This algorithm executes in time within $O(n(1 + \mathcal{H}(v_1, \dots, v_\mu)))$. The merging process is then represented by a tree with the shape of a Huffman [7] tree. Consider the i -th VORONOI DIAGRAM of the input $\forall i \in [1..\mu]$: let c_i be the binary string describing the path leading from the root to the corresponding leaf, and l_i the length of this path. The sum of the computational costs of the binary merges is the sum of the sizes of the VORONOI DIAGRAM computed. The i -th VORONOI DIAGRAM contributes a cost within $O(v_i)$ to l_i levels, which has a sum within $O(\sum_{i=1}^\mu l_i v_i)$. Consider the binary tree where the μ initial VORONOI DIAGRAMS are leaves and the $\mu - 1$ computed VORONOI DIAGRAMS are internal nodes:

- The set of binary strings $\{c_1, \dots, c_\mu\}$ is a prefix free code, i.e. no code is prefix of another root-to-leaf path, simply because they are paths in a tree.
- The lengths of those codes minimize $\sum_{i=1}^\mu l_i r_i$ as a property of Huffman [7] codes.

By the optimality of Huffman codes, this complexity is within a linear term of the entropy of the distribution (v_1, \dots, v_μ) , i.e. $\sum_{i=1}^\mu l_i r_i \in O(n(1 + \mathcal{H}(v_1, \dots, v_\mu)))$. This yields the final time complexity, within $O(n(1 + \mathcal{H}(v_1, \dots, v_\mu)))$. \square

The combination of the partitioning algorithm and the merging process yields an optimal algorithm computing these structures adaptive to the decomposition of the input into monotone histograms.

Theorem 4. *Let d_1 and d_2 be two perpendicular directions. Let S be a sequence of n planar points. Let μ and $\langle v_1, \dots, v_\mu \rangle$ be the minimum number of monotone histograms with respect to d_1 and d_2 and the sizes of these monotone histograms, respectively, in which S can be cut. The DELAUNAY TRIANGULATION and the VORONOI DIAGRAM of S can be computed in time within $O(n(1 + \mathcal{H}(v_1, \dots, v_\mu))) \subseteq O(n(1 + \log \mu)) \subseteq O(n \log n)$, which is worst-case optimal in the comparison model over instances of n points that can be partitioned into monotone histograms of sizes $\langle v_1, \dots, v_\mu \rangle$.*

Proof. The combination of the partitioning algorithm and the merging process yields an algorithm computing these structures within this time. In order to provide a lower bound, we use again the result of $\Omega(n(1 + \mathcal{H}(r_1, \dots, r_\rho)))$ for SORTING a sequence of n numbers, in the worst case over instances covered by ρ runs of lengths r_1, \dots, r_ρ summing n in the comparison model, demonstrated by Barbay and Navarro [2]. This problem can be reduced in linear time to the problem of computing the DELAUNAY TRIANGULATION of a sequence of n planar points covered by ρ monotone histograms of lengths r_1, \dots, r_ρ with respect to the coordinates axes. The ρ runs are transformed into ρ monotone histograms of the same lengths, using points on the parabola, in linear time. The sorted sequence of the numbers can be obtained from the DELAUNAY TRIANGULATION of the points in linear time. \square

Such examples of complete refinements are not the rule: in the following section we describe some examples where such refinements are problematic or impossible.

4 Partial Refinements

We describe a new partitioning algorithm for a sequence of points in Sections 4.1 to 4.3 such that the DELAUNAY TRIANGULATION of each subsequence can be computed in linear time in its length, and discuss alternate partitions in Section 4.4. Each different partitioning algorithm, in combination with the merging process described in Section 3.3, yields a different algorithm adaptive to the input order.

4.1 Incremental Construction

Many Computational Geometry algorithms use tests known as the orientation and incircle tests [18]. The *orientation test* determines whether a point lies to the left of, to the right of, or on a line or plane defined by other points. The *incircle test* determines whether a point lies inside, outside, or on a circle defined by other points.

Green and Sibson [5] proposed the first incremental algorithm for computing the DELAUNAY TRIANGULATION of a point set which finds the triangle containing each new point, and updates the diagram by correcting the edges violating the circumcircle condition—an operation named *flipping*. The algorithm adds points to the structure one by one. For each point, it performs two basic steps:

1. The algorithm finds the triangle containing the new point using the structure as a guide to the relative position of the points. A greedy approach for locating the point is to start at an edge in the structure and to walk across adjacent edges in the direction of the new point until the correct triangle is found. Orientation tests [18] are performed on each edge of such a path to see whether the new point lies on the correct side of that edge. We call the operations involved in this walk *navigating* operations.
2. It then updates the structure adding three new edges from the point inserted to the vertices of the triangle containing the point and flips all invalid edges resulting from the insertion. Note that flipping an edge can make another edge invalid, but that each edge is flipped at most once, so each insertion can trigger at most a linear number of flips.

The time complexity of this incremental algorithm for computing the DELAUNAY TRIANGULATION is within $O(dn) \subseteq O(n^2)$, where $d \in [1..n]$ is an upper bound on the amount of operations required to insert each point and to correct the DELAUNAY TRIANGULATION for the instance being considered. We show the problems that arise when we try to adapt this algorithm in order to obtain a new one whose time complexity is within $O(n \log d) \subseteq O(n \log n)$.

4.2 Adaptivity to D-linear Runs

Consider an incremental algorithm G computing the DELAUNAY TRIANGULATION of n planar points in time within $O(n)$ in the best case; and an algorithm M merging two DELAUNAY TRIANGULATIONS of sizes summing to n in time within $O(n)$ in the worst case. Given a sequence S of n distinct planar points and a constant k , the following naive algorithm computes its DELAUNAY TRIANGULATION in time within $O(n)$ in the best case, in time within $O(n \log n)$ in the worst case, and in time within $O(n(1 + \mathcal{H}(r_1, \dots, r_\rho))) \subseteq O(n \log \rho) \subseteq O(n \log n)$ in the general case, where $\rho \in [1..n]$ and r_1, \dots, r_ρ (such that $n = \sum_{i=1}^{\rho} r_i$) measure the difficulty of partitioning the instance into “easy” subinstances:

1. Run the incremental algorithm G on S until, for the i -th point p , it performs either more than k operations to locate the point on the triangulation, or more than k flip operations on it.
2. Store the DELAUNAY TRIANGULATION of the $i - 1$ first points, and restart the greedy incremental algorithm G on the sequence formed by p and its $n - i$ successors in the input sequence S , until no points are left in S .
3. Let $\rho \in [1..n]$ and r_1, \dots, r_ρ (such that $n = \sum_{i=1}^{\rho} r_i$) be the number of DELAUNAY TRIANGULATIONS computed in this way and the sizes of these DELAUNAY TRIANGULATIONS, respectively. Merge the ρ DELAUNAY TRIANGULATIONS, in overall time within $O(n(1 + \mathcal{H}(r_1, \dots, r_\rho))) \subseteq O(n(1 + \log \rho)) \subseteq O(n \log n)$.

Greedy Partitioning. The algorithm described above suggests a way to partition the input into subsequences such that the DELAUNAY TRIANGULATION of each subsequence can be computed in linear time in its length.

We use the algorithm described to define “easy” sequences of points for the computation of the DELAUNAY TRIANGULATION:

Lemma 5. *For all positive integer n , there exists a sequence of n planar points where the greedy partitioning algorithm yields $\rho \in \Theta(n)$ D-linear runs, and the optimal partition of this sequence has only 2 D-linear runs.*

Proof. Let A be a set of $m > k$ consecutive points in the parabola $y = x^2$ denoted p_1, \dots, p_m from left to right. Let C be the circumcircle of the triangle p_1, p_2, p_3 . Let u, v, w be three points such that the triangle Δuvw with vertex set $\{u, v, w\}$ is small enough with respect to the convex hull of A , and located inside C . Let A' be a D-linear run sequence of l points, of increasing y -coordinate and decreasing x -coordinate, inside C denoted p'_1, \dots, p'_l . During the incremental construction of the DELAUNAY TRIANGULATION of the sequence AA' the first point of A' forces a flipping of m edges in A (i.e. m new edges are created connecting p'_1 with all points in A). Every next point of A' forces a flipping of the m edges connecting points in A with points in A' . Let q'_1 and q'_2 be two consecutive points from the sequence A' and let q_1, \dots, q_{k+1} be $k+1$ consecutive points from A . The sequence $Q = \langle q'_1, q_1, \dots, q_{k+1}, q'_2 \rangle$ is not a D-linear run because q'_2 forces a flipping of the $k+1$ edges connecting q'_1 with all the points in q_1, q_2, \dots, q_{k+1} . Let B be the $(k+3)$ vertex sequence of $k+1$ decreasing area and disjoint triangles so that every two consecutive triangles share an edge. The points u, v and w are the last three points of B . The triangle formed by the first three points of B contains inside the point p_1 of A . B is ordered such that the sequence of vertices is a D-linear run. Let $P = A \cup A' \cup B$ be ordered as $\langle B, p_1, \dots, p_{k+1}, p'_1, p_{k+2}, \dots, p_{2k+3}, p'_2, \dots, p_m \rangle$, that is, first the points of B , then alternate $k+1$ points of A with 1 point of A' , resulting subsequences similar to Q . The greedy partition algorithm divides P into $\min(l, \frac{m}{k+1}) + 1$ D-linear runs: $\langle B \rangle$, $\langle p_1, \dots, p_{k+1} \rangle$ and every subsequence starting with 1 point from A' followed by $k+1$ points from A . Since: (i) adding p_1 to the DELAUNAY TRIANGULATION of B requires traversing k triangles to locate p_1 ; (ii) adding p'_1 to the DELAUNAY TRIANGULATION of $p_1 \dots p_{k+1}$ requires creating k triangles (i.e. p'_1 is adjacent to every point in $p_1 \dots p_{k+1}$); and (iii) adding p'_{i+1} to $\langle p'_i, p_{ik+j}, \dots, p_{(i+1)k+j+1} \rangle$ produces a sequence similar to Q , and hence is not a D-linear run. However, $\langle B \setminus \{u, v, w\} \rangle$ is a D-linear run and $\langle u, v, w, p_1, \dots, p_k, p'_1, p_{k+1}, p_{k+2}, \dots, p_{2k}, p'_2, \dots, p_m \rangle$ is another one. Note that the triangle Δuvw blocks the points in A' of being adjacent to any vertex in A . It is possible to build a sequence $l = \frac{m}{k+1} = \frac{n-k+3}{k+2}$ (1 point in A' for each sequence of $k+1$ points in A) such that the number of D-linear runs yielded by the greedy partitioning algorithm is $\frac{n-k+3}{k+2} + 1$ and the optimal partition has only 2 D-linear runs. \square

4.4 Other Partitioning Schemes

Since the greedy partitioning algorithm does not yield an optimal partition, we explore more sophisticated partition techniques and show that they suffer similar problems.

We adapt the Levcopoulos et al. [11] partition technique to cut a sequence of planar points into D-linear runs. The *test and divide* partitioning is another partition technique based on the incremental algorithm G for the computation of the DELAUNAY TRIANGULATION (seen in Sect. 4.1). Given a sequence $S = \langle p_1, \dots, p_n \rangle$ of n planar points, the *test and divide* partitioning first tests whether the sequence S is a D-linear run. In such a case, it identifies the sequence S as a D-linear run. If not, it partitions the sequence S into $S' = \langle p_1, \dots, p_{\lfloor n/2 \rfloor} \rangle$ and $S'' = \langle p_{\lfloor n/2 \rfloor + 1}, \dots, p_n \rangle$ and recursively the same procedure is applied to S' and S'' . While the greedy partitioning has a linear time complexity, this partitioning has a time complexity within $O(n \log \rho) \subseteq O(n \log n)$ where ρ is the number of D-linear runs identified by the partitioning process.

The following lemma shows that the *test and divide* partitioning suffers the same problems as the greedy partitioning.

Lemma 6. *For all positive integer n , there exists a sequence of n planar points where the test and divide partitioning yields $\rho \in \Theta(n)$ D-linear runs and the optimal partition of this sequence has only 2 D-linear runs.*

Proof. Consider the sequence S used in the proof of Lemma 5 where the number of points in B is changed to $\lfloor n/2 \rfloor$ instead of $k+3$ then the *test and divide* partitioning will cut S in the last point of B . B is a D-linear run, but the bisection partitioning will cut the rest of S in $\frac{n}{2(k+2)}$ D-linear runs. \square

The greedy partitioning algorithm adds the point p_i to the current *run* whether the number of location and update operations of the incremental algorithm G in p_i is at most a threshold k . But it is possible that in some points the number r of operations would be much lower than k . The *amortized greedy* partitioning algorithm makes use of the $k - r$ remaining operations in the subsequent points, similar to the accounting method for amortized analysis. The number $k - r$ of remaining operations are credited to be used later, so that the point p_i will be added to the current run if the number of navigation and update operations of G in p_i is less than k plus this credit.

We use this algorithm to give another definition of “easy” sequences of points for the computation of the DELAUNAY TRIANGULATION:

Definition 4 (Amortized D-linear). Consider a sequence $S = \langle p_1, \dots, p_n \rangle$ of n distinct planar points. Given an integer value $k > 0$, S is Amortized k -D-linear if for each point $p_i \in S$, the incremental algorithm G performs at most $k * i$ navigation and flip operations while computing the DELAUNAY TRIANGULATION of the sequence p_1, \dots, p_i . If S is Amortized k -D-linear and $k \in O(1)$ is a constant independent of n , we say that S is Amortized D-linear.

Such definition yields a partitioning of the input sequence into subsequence of consecutive positions:

Definition 5 (Amortized D-linear Run). An Amortized D-linear run in a sequence S of points is an Amortized D-linear subsequence formed by consecutive points in S .

We define the optimal partition into Amortized D-Linear runs as the partition with the minimum number of Amortized D-linear runs.

Lemma 7. For all positive integer n , there exists a sequence of n planar points where the amortized greedy partitioning yields $\rho \in \Theta(n)$ Amortized D-linear runs and the optimal partition has only 2 Amortized D-linear runs.

Proof. The construction of the sequence follows the same scheme of previous constructions (see the proof of Lemma 5). The sequence B is such that the incremental algorithm G performs k navigation and flip operations in almost every point in B , so that B is still a run but the credit is almost zero at the end of B . Again, the last 3 points of B form a triangle that blocks the points in A' of being adjacent to any vertex in A . The sequence S alternates points in A with points in A' such that each new point in A' is adjacent to every point in A , thus when the numbers of points in A is large enough to exceed the credit, the Amortized D-linear run is cut. This partition algorithm yields close to n/k^2 Amortized D-linear runs. This sequence can be partitioned optimally in just 2 Amortized D-linear runs. Note that the triangle Δuvw blocks the points in A' of being adjacent to any point in A . \square

Amortized Test and Divide Partitioning. The *amortized test and divide partitioning* is a combination of the *test and divide* partitioning and the *amortized greedy* partitioning. Given a sequence $S = \langle p_1, \dots, p_n \rangle$ of n planar points we first test whether the sequence S is Amortized D-linear. In such a case, we identify the sequence S as an Amortized D-linear run. If not, we partition the sequence S into $S' = \langle p_1, \dots, p_{\lfloor n/2 \rfloor} \rangle$ and $S'' = \langle p_{\lfloor n/2 \rfloor + 1}, \dots, p_n \rangle$ and recursively the same procedure is applied to S' and S'' .

Lemma 8. For all positive integer n , there exists a sequence of n planar points where the amortized test and divide partitioning yields $\rho \in \Theta(n)$ Amortized D-linear runs and the optimal partition has only 2 Amortized D-linear runs.

¹⁴*Proof.* The construction of the sequence follows the same scheme of previous constructions (see the proof of Lemma 5). The sequence B is such that the incremental algorithm G performs k navigation and flip operations in almost every point in B , so that B is still a run but the credit is almost zero at the end of B . Again, the last 3 points of B form a triangle that blocks the points in A' of being adjacent to any vertex in A . The sequence S alternates points in A with points in A' such that each new point in A' is adjacent to every point in A , thus when the numbers of points in A is large enough to exceed the credit, the Amortized D-linear run is cut. This partition algorithm yields close to n/k^2 Amortized D-linear runs. This sequence can be partitioned optimally in just 2 Amortized D-linear runs. Note that the triangle Δuvw blocks the points in A' of being adjacent to any point in A . \square

5 Discussion

We describe one technique to refine the analysis of the divide-and-conquer polynomial evaluation algorithm, and two techniques to refine the analysis of divide-and-conquer input-order adaptive algorithms for computing the CONVEX HULL, VORONOI DIAGRAM and DELAUNAY TRIANGULATION of n planar points in time within $O(n(1 + \mathcal{H}(n_1, \dots, n_k))) \subseteq O(n(1 + \log k)) \subseteq O(n \log n)$ for some parameters n_1, \dots, n_k summing to n , where $\mathcal{H}(n_1, \dots, n_k) = \sum_{i=1}^k \frac{n_i}{n} \log \frac{n}{n_i}$ measures the “difficulty” of the instance. We show the optimality of the algorithms for computing CONVEX HULLS, VORONOI DIAGRAMS and DELAUNAY TRIANGULATIONS by providing lower bounds that match the refined analyses. The later results improve on those from Levcopoulos et al. [11] in that this analysis takes advantage of the sizes of the simple polygonal chains into which the input polygonal chain can be partitioned.

There is still work to be done in the directions initiated by our work. One of these techniques is based on the partitioning of a sequence of n planar points into subsequences, whose corresponding VORONOI DIAGRAMS and DELAUNAY TRIANGULATIONS can be computed quickly, and a technique to efficiently merge those structures into a single one, inspired by Huffman [7] codes. Each combination of partition and merging techniques yields adaptive algorithms to the input order for computing VORONOI DIAGRAMS and DELAUNAY TRIANGULATIONS. Those results show that the DELAUNAY TRIANGULATION can be computed in time within $o(n \log n)$ on some classes of instances, yet we described various partitioning algorithms where the time complexity achieved is sub-optimal. The next step is to find better partitioning algorithms taking full advantage of the input order.

We classify the various refined analysis between those focusing on the structure of the instance versus those focusing on the order in which the input is given. We have been analyzing an algorithm for the UNION SET problem that can be adapted for solving the SORTING problem taking advantage of the structure of the instance and the order of the input in synergy. We will continue studying potential synergistic solutions to the SORTING problem and their extension to the CONVEX HULL and DELAUNAY TRIANGULATION computation.

References

1. Afshani, P., Barbay, J., Chan, T.: Instance-optimal geometric algorithms. In: Proceedings 50th IEEE Symposium on Foundations of Computer Science (FOCS) (2009)
2. Barbay, J., Navarro, G.: On compressing permutations and adaptive sorting. Theoretical Computer Science (TCS) 513, 109–123 (2013)
3. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete & Computational Geometry (DCG) 6(5), 485–524 (Aug 1991)
4. Djidjev, H., Lingas, A.: On computing voronoi diagrams for sorted point sets. International Journal of Computational Geometry & Applications (IJCGA) 5(3), 327–337 (1995)
5. Green, P.J., Sibson, R.: Computing dirichlet tessellations in the plane. The Computer Journal (TCJ) 21(2), 168–173 (1978)
6. Guibas, L., Stolfi, J.: Primitives for the manipulation of general subdivisions and the computation of voronoi. ACM Trans. Graph. (TOG) 4(2), 74–123 (Apr 1985)
7. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the Institute of Radio Engineers (IRE) 40(9), 1098–1101 (September 1952)

8. Kirkpatrick, D.G.: Efficient computation of continuous skeletons. In: Proceedings of the 20th Annual Symposium¹⁵ on Foundations of Computer Science (FOCS). pp. 18–27. IEEE Computer Society, Washington, DC, USA (1979)
9. Kirkpatrick, D.G., Seidel, R.: The ultimate planar convex hull algorithm? SIAM Journal on Computing (SICOMP) 15(1), 287–299 (1986)
10. Knuth, D.E.: The Art of Computer Programming, Vol 3, chap. Sorting and Searching, Section 5.3. Addison-Wesley (1973)
11. Levkopoulos, C., Lingas, A., Mitchell, J.S.B.: Adaptive algorithms for constructing convex hulls and triangulations of polygonal chains. In: Proceedings of the 8th Scandinavian Workshop on Algorithm Theory (SWAT). pp. 80–89. Springer-Verlag, London, UK (2002)
12. Melkman, A.A.: On-line construction of the convex hull of a simple polyline. Information Processing Letters (IPL) 25(1), 11–12 (Apr 1987)
13. Moffat, A., Petersson, O.: An overview of adaptive sorting. Australian Computer Journal 24(2), 70–77 (1992)
14. Munro, J.I., Spira, P.M.: Sorting and searching in multisets. SIAM Journal on Computing (SICOMP) 5(1), 1–8 (1976)
15. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer-Verlag (1985)
16. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, New York, NY, USA (1988)
17. Shamos, M., Hoey, D.: Closest point problems. In: Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS). pp. 151–162 (1975)
18. Shewchuk, J.R.: Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. Discrete & Computational Geometry 18(3), 305–363 (Oct 1997)
19. Strassen, V.: Gaussian elimination is not optimal. Numerische Mathematik 13(4), 354–356 (1969)